10/031226
531 Rec'd PCT/PTO 17 JAN 2002

PROCEDE DE SECURISATION D'UN LANGAGE DU TYPE A DONNEES TYPEES, NOTAMMENT DANS UN SYSTEME EMBARQUE ET SYSTEME EMBARQUE DE MISE EN ŒUVRE DU PROCEDE

L'invention concerne un procédé de sécurisation dynamique d'un langage du type à données typées, notamment pour un système embarqué à puce électronique.

L'invention concerne encore un système embarqué à puce électronique pour la mise en œuvre du procédé.

Dans le cadre de l'invention, le terme "système embarqué" doit être compris dans son sens le plus général. Il concerne notamment toutes sortes de terminaux légers munis d'une puce électronique, et plus particulièrement les cartes à puce proprement dites. La puce électronique est munie de moyens d'enregistrement et de traitement de données numériques, par exemple un microprocesseur pour ces derniers moyens.

Pour fixer les idées, et sans que cela limite en quoi que ce soit sa portée, on se placera ci-après dans le cas de l'application préférée de l'invention, à savoir les applications à base de cartes à puce, sauf mention contraire.

De même, bien que divers langages informatiques, tels les langages "ADA" ou "KAMEL" (tous deux étant des marques déposées), sont du type dit à données ou objets typés, un des langages les plus utilisés dans le domaine préféré de l'invention étant le langage de type objet "JAVA" (marque déposée), ce langage sera pris comme exemple ci-après, pour décrire en détail le procédé de l'invention.

Enfin, le terme "sécurisation" doit lui aussi être compris dans un sens général. Notamment, il concerne aussi bien ce qui a trait au concept de confidentialité des données manipulées qu'au concept d'intégrité, matérielle et/ou logicielle, des composants présents dans le système embarqué.

Avant de décrire plus avant l'invention, il est tout d'abord utile de rappeler brièvement les principales caractéristiques du langage "JAVA", notamment dans un environnement du type carte à puce.

Ce dernier langage présente en particulier l'avantage d'être multiplateformes : il suffit que la machine dans laquelle s'exécute l'application écrite en langage "JAVA" soit munie d'un minimum de ressources informatiques spécifiques, notamment d'une pièce de logiciel appelé "machine virtuelle JAVA" pour l'interprétation d'une suite de séquences d' "opcodes" d'instructions de 8 bits, appelées "bytecode" ou "p-code" (pour "program code" ou code programme). Le "p-code" est enregistré dans des positions de mémoire des moyens d'enregistrement de données précités. Plus précisément, dans le cas du langage "JAVA", la zone occupée par les positions de mémoire, d'un point de vue logique, se présente sous une configuration connue sous le nom de pile.

Dans le cas d'une carte à puce, celle-ci intègre la "machine virtuelle JAVA" (enregistrée dans ses moyens de mémoire) et fonctionne en interprétant un langage basé sur la séquence d'opcodes précitée. Le code exécutable ou "p-code" résulte d'une compilation préalable. Le compilateur est agencé pour que le langage transformé obéisse à un format déterminé et respecte un certain nombre de règles fixées *a priori*.

Les "opcodes" peuvent recevoir des valeurs d'éléments les suivants dans une séquence du "p-code", ces éléments sont alors appelés paramètres. Les opcodes peuvent aussi recevoir des valeurs en provenance de la pile. Ces éléments constituent alors des opérandes.

Selon une autre caractéristique du langage "JAVA", il est mis en œuvre des éléments connus sous les noms de "classes" et de "méthodes". Lors de l'exécution d'une méthode donnée, la machine virtuelle retrouve le "p-code" correspondant. Ce "p-code" identifie des opérations spécifiques à effectuer par la machine virtuelle. Une pile particulière est nécessaire pour le traitement de variables dites locales, d'opérations arithmétiques ou pour l'invocation d'autres méthodes.

Cette pile sert de zone de travail pour la machine virtuelle. Pour optimiser les performances de la machine virtuelle, la largeur de la pile est généralement fixée pour un type primitif donné.

Dans cette pile deux grands types d'objets peuvent être manipulés :

- des objets de type dit "primitif", ceux connus sous les dénominations "*int*" (pour entier long : 4 octets), "*short*" (pour entier court : 2 octets), "*byte*" (octet), "*boolean*" (objet booléen) ; et
- des objets de type dit "référence" (tableaux d'objets de type primitif, instances de classes).

La différence fondamentale entre ces deux types d'objets est que seule la machine virtuelle attribue une valeur à des objets de type référence et les manipule.

Les objets références peuvent être vus comme des pointeurs vers des zones mémoires de la carte à puce (références physiques ou logiques).

Le langage "JAVA", dont les principales caractéristiques viennent d'être succinctement rappelées, se prête particulièrement bien aux applications mettant en jeu des interconnexions avec le réseau Internet et son grand succès est d'ailleurs lié au fort développement des applications Internet.

D'un point de vue sécurité, il présente aussi un certain nombre d'avantages. Tout d'abord, le code exécutable ou "p-code" résulte d'une compilation préalable. Le compilateur peut donc être agencé, comme il a été indiqué, pour que le langage transformé obéisse à un format déterminé et respecte un certain nombre de règles fixées *a priori*.

Une de ces règles est qu'une application donnée soit confinée à l'intérieur de ce qui est appelé une "sand box" (ou "boite noire"). Les instructions et/ou données associées à une application déterminée sont mémorisées dans des positions de mémoire des moyens d'enregistrement de données. Dans le cas du langage "JAVA", d'un point de vue logique, la configuration de ces moyens d'enregistrement de données prend la forme d'une pile. Le confinement dans une "sand box" signifie en pratique que les

instructions précitées ne peuvent pas adresser des positions mémoires en dehors de celles affectées à ladite application, sans y être autorisées expressément.

Cependant, une fois chargé en mémoire, des problèmes de sécurité peuvent se poser si le "p-code" a été altéré ou si son format n'a pas respecté les spécifications de la machine virtuelle. Aussi, dans l'art connu, notamment lorsqu'il s'agit d'applications, par exemple des "applets" (appliquettes), téléchargées via le réseau Internet, le code compilé, c'est-à-dire le "p-code" est vérifié par la machine virtuelle. Cette dernière est habituellement associée à un navigateur de type "WEB" dont est muni le terminal connecté au réseau Internet. Pour ce faire, la machine virtuelle est elle-même associée à une pièce de logiciel particulière ou vérificateur.

Cette vérification peut s'effectuer en mode dit "off-line", c'est-à-dire hors connexion, ce qui ne pénalise pas le traitement de l'application, notamment d'un point de vue coût de communication.

On est ainsi sûr, après que la vérification soit effectuée, que le "p-code" n'est pas endommagé et est conforme au format et aux règles préétablis. On est aussi sûr, dans ces conditions, que lors de l'exécution du "p-code", il n'y aura pas de détérioration du terminal dans lequel il s'exécute.

Cependant, ce procédé n'est pas sans inconvénients, en particulier dans le cadre des applications visées préférentiellement par l'invention.

Tout d'abord, le vérificateur précité nécessite à lui seul une quantité de mémoire relativement importante, de l'ordre de plusieurs MO. Cette valeur élevée ne présente pas de problèmes particuliers si le vérificateur est enregistré dans un micro-ordinateur ou un terminal similaire disposant de ressources mémoires élevées. Cependant, lorsque l'on envisage d'utiliser un terminal de traitement de données possédant des ressources informatiques plus limitées, *a fortiori* une carte à puce, il n'est pas envisageable, d'un point de vue pratique, compte tenu des technologies actuellement disponibles, d'implémenter le vérificateur dans ce type de terminal.

On doit également noter que la vérification est d'un type que l'on peut qualifier de "statique", car effectuée une fois pour toute, avant l'exécution du "p-code". Lorsqu'il s'agit d'un terminal du type micro-ordinateur, notamment lorsque ce dernier est maintenu déconnecté lors de l'exécution du "p-code" vérifié au préalable, cette dernière caractéristique ne pose pas de problèmes particuliers. En effet, il n'existe pas de risques importants, d'un point de vue sécurité, car le terminal reste habituellement sous le contrôle de son opérateur.

Tel n'est pas le cas pour un système embarqué mobile, notamment pour une carte à puce. En effet, si le "p-code", même vérifié, est ensuite chargé dans les moyens d'enregistrement de données de la carte à puce, il peut subir *a posteriori* des altérations. En général, la carte à puce, ce par nature, n'est pas destinée à demeurer en permanence dans le terminal à partir duquel l'application a été chargée. A titre d'exemple non limitatif, la carte à puce peut être soumise à un rayonnement ionisant qui altère physiquement des positions de mémoire. Il est possible également d'altérer le "p-code" au moment de son téléchargement dans la carte à puce, à partir du terminal.

Il s'ensuit que, si le "p-code" est altéré, notamment dans un but malveillant, il est possible d'effectuer une opération dite de "dump" (duplication) de zones de mémoires et/ou de mettre en péril le bon fonctionnement de la carte à puce. Il devient ainsi possible, par exemple, et malgré la disposition dite de "sand box" précitée, d'avoir accès à des données confidentielles, ou pour le moins non autorisées, ou d'attaquer l'intégrité d'une ou plusieurs applications présentes sur la carte à puce. Enfin, si la carte à puce est connectée au monde extérieur, les dysfonctionnements provoqués peuvent se propager à l'extérieur de la carte à puce.

L'invention vise à pallier les inconvénients des procédés et dispositifs de l'art connu, et dont certains viennent d'être rappelés.

L'invention se fixe pour but un procédé de sécurisation dynamique d'applications en langage du type à données typées dans un système embarqué.

Elle se fixe également pour but un système pour la mise en œuvre de ce procédé.

Pour ce faire, selon une première caractéristique, un élément d'information binaire comprenant un ou plusieurs bits, que l'on appellera ci-après "élément d'information de type", est associé à chaque objet manipulé par la machine virtuelle, dans le cas du langage "JAVA" précité. De façon plus générale, un élément d'information de type est associé à chaque donnée typée manipulée dans un langage donné, du type à objets ou données typés.

Selon une autre caractéristique, les éléments d'information de type sont stockés physiquement dans des zones de mémoire particulières des moyens de mémorisation du système embarqué à puce électronique.

Selon une autre caractéristique encore la machine virtuelle, toujours dans le cas du langage "JAVA" vérifie lesdits éléments d'information de type lors de certaines opérations d'exécution du "p-code", telles la manipulation d'objet dans la pile, etc., opérations qui seront précisées ci-après. De façon plus générale également, pour un autre langage, le processus est similaire et il est procédé à une étape de vérification des éléments d'information de type. On constate donc que, de façon avantageuse, ladite vérification est d'un type que l'on peut appeler dynamique, puisqu'effectuée en temps réel lors de l'interprétation ou de l'exécution du code.

La machine virtuelle, ou ce qui en tient lieu pour un langage autre que le langage "JAVA", vérifie, en continu et avant ladite exécution d'une instruction ou d'une opération, que l'élément d'information de type correspond bien au type attendu de l'objet ou de la donnée typé à manipuler. Lorsqu'un type incorrect est détecté, des mesures sécuritaires sont prises afin de protéger la machine virtuelle et/ou d'empêcher toutes opérations non

conformes et/ou dangereuses pour l'intégrité du système embarqué à puce électronique.

Selon une première variante de réalisation supplémentaire du procédé selon l'invention, lesdits éléments d'information de type sont également utilisés avantageusement pour permettent la gestion de piles de largeurs variables, ce qui permet d'optimiser l'espace mémoire du système embarqué à puce électronique, dont les ressources de ce type sont, par nature, limitées, comme il a été rappelé.

Selon une deuxième variante de réalisation supplémentaire, cumulable avec la première, les éléments d'information de type sont également utilisés, en y adjoignant un ou plusieurs bit(s) d'information supplémentaire(s), utilisés comme "drapeau" ("flags" selon la terminologie anglo-saxonne), pour marquer les objets ou les données typées. Ce marquage est alors utilisé pour indiquer si ces derniers éléments sont utilisés ou non, et dans ce dernier cas, s'ils peuvent être effacés de la mémoire, ce qui permet également de gagner de la place mémoire.

L'invention a donc pour objet principal un procédé pour l'exécution sécurisée d'une séquence d'instructions d'une application informatique se présentant sous la forme de données typées enregistrées dans une première série d'emplacements déterminés d'une mémoire d'un système informatique, notamment un système embarqué à puce électronique, caractérisé en ce que des données supplémentaires dites éléments d'information de type sont associés à chacune desdites données typées, de manière à spécifier le type de ces données, en ce que lesdits éléments d'information de type sont enregistrés dans une deuxième série d'emplacements de mémoire déterminés de ladite mémoire de système informatique, et en ce que, avant l'exécution d'instructions d'un type prédéterminé, il est procédé à une vérification en continu, préalable à l'exécution d'instructions prédéterminées, de la concordance entre un type indiqué par ces instructions et un type attendu indiqué par lesdits éléments d'information de type enregistrés dans ladite deuxième série d'emplacement

de mémoire, de manière n'autoriser ladite exécution qu'en cas de concordance entre lesdits types.

L'invention a encore pour objet un système embarqué à puce électronique pour la mise en œuvre de ce procédé.

L'invention va maintenant être décrite de façon plus détaillée en se référant aux dessins annexés, parmi lesquels :

- les figures 1A à 1G illustrent les principales étapes d'une exécution correcte d'un exemple de "p-code" dans une mémoire à pile associée à des zones de mémoire spécifiques stockant des données dites éléments d'information de type selon l'invention ;
- les figures 2A et 2B illustrent schématiquement des étapes d'exécution de ce même code, mais contenant une altération menant à une exécution incorrecte et une détection de cette altération par le procédé de l'invention ; et
- la figure 3 illustre schématiquement un système comprenant une carte à puce pour la mise en œuvre du procédé selon l'invention.

Dans ce qui suit, sans en limiter en quoi que ce soit la portée, on se placera ci-après dans le cadre de l'application préférée de l'invention, sauf mention contraire, c'est-à-dire dans le cas d'un système embarqué à puce électronique intégrant une machine virtuelle "JAVA" pour l'interprétation de "p-code".

Comme il a été rappelé dans le préambule de la présente description, lors de l'exécution d'une méthode donnée, la machine virtuelle retrouve le "p-code" correspondant. Ce "p-code" identifie des opérations spécifiques à effectuer par la machine virtuelle. Une pile particulière est nécessaire pour le traitement de variables dites locales, d'opérations arithmétiques ou pour l'invocation d'autres méthodes.

La pile sert de zone de travail pour la machine virtuelle. Pour optimiser les performances de la machine virtuelle, la largeur de la pile est généralement fixée pour un type primitif donné.

Comme il a été également rappelé, dans cette pile deux grands types d'objets peuvent être manipulés :

- des objets de type dit "primitif", ceux connus sous les dénominations "*int*" (pour entier long : 4 octets), "*short*" (pour entier court : 2 octets), "*byte*" (octet), "*boolean*" (objet booléen) ; et
- des objets de type dit "référence" (tableaux d'objets de type primitif, instances de classes).

C'est ce dernier type d'objets qui pose le plus de problème, d'un point de vue sécurité, puisqu'il existe des possibilités, comme indiqué précédemment, de les manipuler de façon artificielle et de créer ainsi des dysfonctionnements de natures diverses.

Ils existent plusieurs types d' "opcodes", et notamment :

- la création d'un objet de type primitif (par exemple les opcodes dénommés "*bipush*" ou "*iconst*") ;
- l'exécution d'opérations arithmétiques sur des objets de type primitif (par exemple les "opcodes" dénommés "*iadd*" ou "*sadd*") ;
- la création d'un objet référence (par exemple les "opcodes" dénommés "*new*", "*newarray*" ou "*anewarray*").
- la gestion de variables locales (par exemple les "opcodes" dénommés "*aload*", "*iload*" ou "*istore*") ; et
- la gestion de variables de classes (par exemple les "opcodes" dénommés "*getstatic_a*" ou "*putfield_i*").

Chaque "opcode" qui utilise des objets placés en pile est typé afin de s'assurer que son exécution puisse être contrôlée. Généralement la(les) première(s) lettre(s) des "opcodes" indique(nt) le type utilisé. A titre d'exemple, et pour fixer les idées, (la ou les première(s) lettre(s) étant graissée pour mettre en évidence cette disposition), on peut citer les "opcodes" suivants :

- "***a**load*" pour les objets références ;
- "***i**load*" pour les entiers ; et
- "***ia**load*" pour les tableaux d'entiers.

Dans ce qui suit, par mesure de simplification la "machine virtuelle JAVA" sera appelée JVM.

Selon une première caractéristique du procédé selon l'invention, des éléments d'information de type sont stockés dans une zone mémoire sous la forme, chacun, d'un ou de plusieurs bits. Chacun de ces éléments d'information de type caractérise un objet manipulé par la JVM. On associe notamment un élément d'information de type à :

- chaque objet empilé dans la zone de donnée de la pile ;
- chaque variable locale (variable dont la portée ne dépasse pas le cadre d'une méthode) ; et
- à chaque objet de ce qui est appelé le "heap", c'est-à-dire une zone de mémoire stockant les objets dits "référence", chaque tableau et chaque variable globale.

Cette opération peut être appelée "typage" des objets. Selon une deuxième caractéristique du procédé de l'invention, la JVM vérifie le typage dans les cas suivants :

- lorsqu'un "opcode" manipule un objet stocké dans la pile ;
- récupère un objet dans la zone du "heap" ou dans celle des variables locales pour le placer en pile ;
- modifie un objet dans la zone du "heap" ou dans celle des variables locales ; et
- lors de l'invocation d'une nouvelle méthode, lorsque les opérandes sont comparés à la signature de la méthode.

Selon une autre caractéristique du procédé de l'invention, la JVM vérifie, avant l'exécution des opérations ci-dessus, que leurs types correspondent bien à celui attendu (c'est-à-dire ceux donnés par les "opcode" à effectuer).

Dans le cas de la détection d'un type incorrect, des mesures sécuritaires sont prises afin de protéger la JVM et/ou d'empêcher toutes opérations illégales ou dangereuses pour l'intégrité du système, tant d'un point de vue logique que matériel.

Pour mieux expliciter le procédé selon l'invention, on va maintenant le détailler en considérant un exemple particulier de code source en langage "JAVA".

On suppose également que la JVM est associée à une pile de 32 bits comportant au plus 32 niveaux et supportant les types primitifs (par exemple "*int*", "*short*", "*byte*", "*boolean*" et "*object reference*")

Le typage de la pile, selon l'une des caractéristiques de l'invention, peut alors être réalisé à l'aide d'éléments d'information de type de longueur 3 bits, conformément à la TABLE I placée en fin de la présente description. Les valeurs portées dans la TABLE I sont naturellement arbitraires. D'autres conventions pourraient être prises sans sortir du cadre de l'invention.

Le code source "JAVA" qui va être considéré ci-après à titre d'exemple particulier est le suivant :

Source "JAVA" (1) :

```
Public void method(){
int[] buffer;                 //Déclaration
buffer=new int[2] ;           // création d'un tableau d'entiers de 2
éléments
buffer[1]=5 ;                 // initialisation du tableau avec la valeur 5
}
```

Après un passage dans un compilateur approprié, un fichier "classe" contenant le "p-code" (2) correspondant au code source ci-dessus (1) est obtenu. Il se présente comme suit :

"p-code" (2) :

```
iconst_2    // Push int constant 2
newarray    T_INT
```

```
astore_1    int[] buffer
aload_1     int[] buffer
iconst_1    // Push int constant 1
iconst_5    // Push int constant 5
iastore
return
```

Comme il est bien connu de l'homme de métier, les trois premières lignes correspondent à la création du tableau précité (voir code source (1)). Les cinq dernières lignes correspondent à l'initialisation de ce tableau

On va maintenant illustrer en détail les étapes d'une exécution correcte du "p-code" ci-dessus. Puisque le "p-code" est un langage de type interprété, les lignes successives sont lues les unes après les autres et les étapes précitées correspondent à l'exécution de ces lignes, avec éventuellement l'exécution d'itérations et/ou de branchements. Dans ce qui suit, les différentes lignes de code sont graissées pour les mettre en évidence.

Exécution correcte :

Etape 1 : ***"iconst_2"***

La figure 1A illustre de façon schématique l'étape d'exécution de ce "p-code". On a représenté, sous la référence 1, la mémoire du système embarqué à puce électronique (non représenté). De façon plus précise, cette mémoire 1 est divisée en quatre parties principales, deux étant communes à l'art connu : la zone dite "*zone data*" (données) 2*a* et la zone dite "*zone variable locale*" 3*a*. Ces zones, 2*a* et 3*a*, constituent la pile proprement dite de la machine virtuelle "JAVA" (JVM) que l'on appellera ci-après par simplification "*pile de la JVM*".

A ces zones sont associées des zones de mémoire, 4*a* et 5*a*, respectivement, spécifiques à l'invention, que l'on appellera zones de

"*Typage*". Selon un des aspects de l'invention, les zones de mémoire, 4*a* et 5*a*, sont destinées à stocker des éléments d'information de type (de longueur 3 bits dans l'exemple décrit) associés aux données stockées dans les zones 2*a* et 3*a*, respectivement, dans des emplacements de mémoire en relation biunivoque avec les emplacements de mémoire de ces zones. L'organisation logique de ces zones de mémoire est du type dit "pile" comme rappelé. Aussi, elles ont été représentées sous la forme de tableaux de dimensions *c*x*l*, avec *c* nombre de colonnes et *l* nombre de lignes, c'est-à-dire la "hauteur" de la pile ou niveau (qui peut varier à chaque étape de l'exécution d'un "p-code"). Dans l'exemple, *c*=4 pour les zones "*zone data*" 2*a* et "*zone variable locale*" 3*a* (chaque colonne correspondant à une position de mémoire de 4 octets, soit au total 32 bits), et *c*=3 pour les zones de "*typage*", 4*a* et 5*a*, (chaque colonne correspondant à une position de mémoire de 1 bit). Sur la figure 1A, le nombre de lignes représenté (ou numéro de niveau : 1 à 32 maximum dans l'exemple décrit) est égal à 2 pour toutes les zones de mémoire. Chacune des zones de mémoire, 2*a* à 5*a*, constitue donc une pile élémentaire.

On doit bien comprendre cependant que, physiquement, les positions de mémoires précitées peuvent être réalisées à base de divers circuits électroniques : cellules de mémoire vive, registres, etc. De même, elles ne sont pas forcément contiguës dans l'espace mémoire 1. La figure 1A ne constitue qu'une représentation schématique de l'organisation logique en piles de la mémoire 1.

L' "opcode" à exécuter pendant cette première étape n'a ni paramètre, ni opérande. La valeur entière 2 (soit "0002") est placée dans la pile : au niveau 1 (ligne inférieure dans l'exemple) de la zone 2*a*. La zone de "*Typage*" correspondante 4*a* est mise à jour.

D'après les conventions de la TABLE I, la valeur "*int*" (entier) "000" (en bits) est placée dans la zone de "*Typage*" 4*a*, également au niveau 1 (ligne inférieure). Aucune valeur n'est placée dans la "*zone variable locale*" 3*a*. Il en est de même de la zone de "*Typage*" correspondante 5*a*.

Etape 2 : **newarray T_INT**

L'étape correspondante est illustrée par la figure 1B.

Les éléments communs à la figure 1A portent les mêmes références numériques et ne seront re-décrits qu'en tant que de besoin. Seule la valeur littérale associée aux valeurs numériques est modifiée. Elle est identique à celle de la figure correspondante, soit *b* dans le cas de la figure 1B, de manière à caractériser les modifications successives des contenus des zones de mémoire. Il en sera de même pour les figures suivantes 1C à 1G.

L' "opcode" à exécuter pendant cette deuxième étape a pour paramètre le type de tableau à créer (soit type "*int*").

Cet "opcode" a pour opérande une valeur qui doit être de type "*int*", correspondant à la taille du tableau à créer (soit 2).

La vérification de la zone de "Typage" (à l'état 4*a*) indique un type correct. L'exécution est donc possible.

Un objet référence est créé dans la "Pile JVM" : par exemple la valeur (arbitraire) de quatre octets "1234" est placée dans les positions de mémoire de la "*zone variable locale*" (niveau 1). Puisqu'il s'agit d'un objet de type référence, la valeur "100" (en bits) est placée dans la zone de "Typage" correspondante 5*b* (niveau 1).

Aucune valeur n'est placée dans la zone de mémoire 3*b*, ni dans la zone de "*Typage*" 5*b*.

Etape 3 : **astore_1 int[] buffer**

Cette étape est illustrée par la figure 1C.

L' "opcode" a pour opérande une valeur qui doit être de type "Objet référence". La vérification de la zone de "Typage" (à l'état 4*b*) indique un type correct. L'exécution est donc possible.

L'objet référence est déplacé vers la "*zone variable locale*" 3*c* : emplacement 1 (niveau 1).

Les zones de "Typage", 4*c* et 5*c* sont mise à jour : la valeur "100" (en bits) est déplacée du niveau 1 de la zone 4*c* vers le niveau 1 de la zone 5*c*.

Etape 4 : **aload_1 int[] buffer**

Cette étape est illustrée par la figure 1D.

Cet "opcode" a pour objet d'empiler l'objet référence "1234", stocké dans la "*zone variable locale*" 3*d*, au niveau 1 de la "*zone data*" 2*d*, c'est-à-dire dans les positions de mémoire de la ligne inférieure de cette zone.

La vérification de la zone de "Typage" (à l'état 5*c*) indique un type correct. L'exécution est donc possible.

L'objet référence "1234" est placé dans la "*zone data*" 2*d*.

Les zones de "Typage" 4*d* et 5*d* sont mises à jour et stockent toutes deux, dans les emplacements de mémoire correspondants, la valeur "100" (en bits), représentative d'un type "Objet référence".

Etape 5 : **iconst_1 // Push int constant 1**

Cette étape est illustrée par la figure 1E.

L' "opcode" à exécuter pendant cette étape n'a ni paramètre ni opérande. La valeur entière 1 (soit "0001") est placée dans la pile : emplacement 2 (niveau 2) de la "*zone data*" 2*e*. La zone de "*Typage*" correspondante 4*e* est mise à jour, également au niveau 2 (le niveau 1 reste inchangé : valeur "1000"). La valeur "*int*" (entier) "000" (en bits) est placée dans la zone de "*Typage*" 4^e (niveau 2). Les zones 3*e* et 5*e* restent inchangées.

Etape 6 : **iconst_5 // Push int constant 5**

Cette étape est illustrée par la figure 1F.

L' "opcode" à exécuter pendant cette étape n'a ni paramètre ni opérande. La valeur entière 1 (soit "0001") est placée dans la pile : niveau 3 de la "*zone data*" 2*f*. La zone de "*Typage*" correspondante 4*f* est mise à jour, également au niveau 3 (les niveaux 1 et 2 restent inchangés : valeurs "1000" et "000" respectivement). La valeur "*int*" (entier) "000" (en bits) est placée dans la zone de "*Typage*" 4*f*. Les zones 3*f* et 5*f* restent inchangées.

Etape 7 : **iastore**

Cette étape est illustrée par la figure 1G.

Cet "opcode" a pour opérande une valeur de type "*int*", un index de type "*int*" et un objet référence de type tableau.

La vérification de la zone de "*Typage*" (à l'état 4*f* : niveau 3) indique un type correct. L'exécution est donc possible.

La valeur est stockée dans l'objet référence à l'index donné.

Etape 7 : **return**

Cet "opcode" indique la fin de la méthode, la pile doit alors être vide.

En considérant de nouveau le même "p-code" (voir (2), obtenu après compilation du code source (1)), on va maintenant détailler un exemple d'exécution incorrecte.

Exécution incorrecte :

A l'étape que l'on nommera 4' (correspondant à l'étape 4 : figure 1D). Il est supposé que le "p-code" a été altéré et que l' "opcode" :

"aload_1 int [] buffer" ,

a été remplacé, par exemple, par l' "opcode" suivant :

"iipush 0x5678",

instruction dans laquelle " 0x" indique une valeur hexadécimale.

Comme illustré par la figure 2A, cet "opcode", de type objet de référence, stocké au niveau 1 de la "*zone variable locale*" 3*a*', a pour objet d'empiler un entier de valeur "5678" dans la pile, dans la "*zone data*" 2'*a*.

La zone de "*Typage*" 4a' va être mise à jour. Il s'ensuit que les niveaux 1 des zones de "*Typage*", 4*a*' et 5*a*', vont tous deux contenir la valeur "100" (en bits), c'est-à-dire une valeur associée à un "Objet référence". Cette configuration particulière est illustrée par la figure 2A.

L'exécution se poursuit normalement comme dans le cas précédemment illustré par référence aux figures 1E et 1F.

Etape 5' : **iconst_1 // Push int constant 1**

Etape 6' : **iconst_5 // Push int constant 5**

L'état des zones de la "*pile de la JVM*", "*zone variable locale*" 3b' et "*zone data*" 2b', est illustré par la figure 2B. de façon plus précise la "*zone data*" 2b' enregistre, au niveau 1, la valeur entière "5678", au niveau 2, la valeur entière "0001" et au niveau 3, la valeur entière "0005". La "*zone variable locale*" 3a' est restée inchangée. Il en est de même de la zone de "*Typage*" correspondante 5*a*'. Par contre, la zone de "*Typage*" 4*b*' est mise à jour et les valeurs suivantes sont enregistrées aux niveaux respectifs 1 à 3 : "100", "000" et "000" (en bits).

Etape 7' : **iastore**

Cet "opcode" a pour opérande une valeur de type "*int*", un index de type "*int*" et un objet référence de type tableau.

La vérification de la zone de "*Typage*" (niveau 1 de la zone, à l'état 4*b*') indique que le code détecté est incorrect. En effet, un entier ("*int*" ; code "000") est attendu à la place d'un "Objet référence" (code "100").

La JVM détecte donc la présence d'un "opcode" illégal menaçant la sécurité du système. L'exécution normale de la séquence d'instructions en cours est interrompue et remplacée par l'exécution d'instructions correspondant à des mesures sécuritaires pré-programmées : signal d'alerte, etc.

On a supposé jusqu'à présent que la largeur (ou taille) de la "*pile de la JVM*" ; que ce soit celle de la "*zone data*" ou la "*zone variable locale*", était fixe, ce qui est généralement le cas dans l'art connu. Dans l'exemple décrit, on a supposé que chaque emplacement de mémoire compte quatre octets (soit 32 bits). Cependant, une telle disposition s'avère pénalisante en terme de capacité de mémoire. En effet, d'une application logicielle à l'autre, voire à l'intérieur d'une même application, le nombre d'octets nécessaire pour chaque instruction est variable. Comme il a été indiqué, l'agencement les piles élémentaires des "*zone data*" et "*zone variable locale*" telles qu'illustrées par les figures 1A à 1G, ou 2A à 2B, ne représentent qu'une vue logique de l'espace mémoire 1. Il est donc tout à fait possible de conserver

une architecture logique du type pile, même si les emplacements de mémoire, successifs ou non, sont de longueurs variables, voire même si les différentes positions (cellules) de mémoire sont physiquement dispersées.

Aussi, selon une première variante supplémentaire du procédé selon l'invention, les éléments d'information de type permettent aussi de déterminer la largeur instantanée nécessaire, en positions de mémoire, des zones de la "*pile de la JVM*". Il suffit, pour ce faire, que les codes enregistrés dans les zones de "*Typage*" de la mémoire soient associés, en tout ou partie, à une information caractérisant la largeur de la pile précitée. A titre d'exemple non limitatif, il peut s'agir de bits supplémentaires, ajoutés aux codes de typage, ou d'une combinaison de bits non utilisée de ces codes. Dans le premier cas, si la largeur de la pile peut varier, toujours à titre d'exemple, entre 1 et 4 octets, il suffit de 2 bits supplémentaires pour caractériser les largeurs suivantes :

Configuration binaire	00	01	10	11
Largeur en octets	1	2	3	4

Cette disposition, qui permet d'optimiser l'espace mémoire en fonction des applications à exécuter, conduit à un gain de place de mémoire substantiel, ce qui constitue un avantage appréciable lorsqu'il s'agit de dispositifs, telle notamment une carte à puce, dont les ressources de stockage sont limitées par nature.

Selon une deuxième variante de réalisation du procédé selon l'invention, il est également possible d'utiliser les éléments d'information de type pour indiquer si un objet est encore utilisé (c'est-à-dire doit être conservé) ou peut être effacé de la "*zone variable locale*". En effet, au bout d'un certain nombre d'opérations, un objet donné enregistré dans cette zone n'est plus utilisé. Le laisser en permanence constitue donc une perte inutile d'espace mémoire.

A titre d'exemple non limitatif, on peut ajouter un bit d'information aux codes enregistrés dans les zones de "*Typage*", faisant fonction de drapeau, ou "*flag*" selon la terminologie anglo-saxonne. L'état de ce bit indique alors si l'objet doit être conservé (car encore utilisé) ou peut être effacé, et le marque comme tel. Les conventions arbitraires suivantes peuvent être adoptes :

- état logique "0" = objet utilisé
- état logique "1" = objet pouvant être effacé

Cette disposition, que l'on peut qualifier de mécanisme de type "garbage collector" (ou "ramasse-miettes") permet aussi un gain en espace mémoire.

Naturellement, les dispositions propres aux deux variantes de réalisation supplémentaires qui viennent d'être décrites peuvent être cumulées.

La figure 3 illustre schématiquement un exemple d'architecture de système informatique à base d'applications de carte à puce pour la mise en œuvre du procédé selon l'invention qui vient d'être décrit.

Ce système comprend un terminal 7, qui peut être relié ou non à des réseaux extérieurs, notamment au réseau Internet *RI*, par un modem ou tous moyens équivalents 71. Le terminal 7, par exemple un micro-ordinateur, comprend notamment un compilateur 9. Le code peut être compilé à l'extérieur du terminal pour donner un fichier dit " Class" (compilateur "JAVA" vers "Class"), c'est ce fichier qui est téléchargé par un navigateur Internet, le micro-ordinateur comprend lui un convertisseurr qui donne un fichier dit "Cap" ("Class" vers "Cap"). Ce convertisseur réduit notamment la taille du fichier "Class" pour permettre de le charger sur une carte à puce. Une application quelconque, par exemple téléchargée via le réseau Internet *RI* et écrite en langage "JAVA" est compilée par le compilateur 9 et chargée, via un lecteur de carte à puce 70 dans les circuits de mémoire 1 de la carte à puce 8. Celle-ci intègre, comme il a été rappelé, une machine virtuelle "JAVA (JVM) 6 capable d'interpréter le "p-code" issu de la compilation et

chargés dans la mémoire 1. On a également représenté différentes piles de mémoire : les zones "*zone data*" 2 et "*zone variable locale*" 3, ainsi que les zones de typage, 4 et 5, ces dernières spécifiques à l'invention. La carte à puce 8 comprend également des moyens classiques de traitement de données reliés à la mémoire 1, par exemple un microprocesseur 80.

Les communications entre la carte à puce 8 et le terminal 7, via le lecteur 70, d'une part, et entre le terminal 7 et le monde extérieur, par exemple le réseau Internet *RI*, via le modem 71, d'autre part, s'effectuent de façon également classique en soi, et il n'y pas lieu de les décrire plus avant.

A la lecture de ce qui précède, on constate aisément que l'invention atteint bien les buts qu'elle s'est fixés.

Elle permet une exécution sécurisée d'une suite d'instructions d'une application écrite langage du type à données typées se déroulant dans une mémoire à architecture de type pile. Le degré de sécurisation élevé est obtenu notamment du fait que la vérification du code est effectuée de façon dynamique, selon un des aspects de l'invention.

Cette disposition permet en outre, au prix d'une augmentation minime du temps de traitement, de se passer d'un vérificateur nécessitant des ressources de mémoire importantes. Ce type de vérificateur ne peut d'ailleurs convenir, dans la pratique, aux applications préférées de l'invention.

Il doit être clair cependant que l'invention n'est pas limitée aux seuls exemples de réalisations explicitement décrits, notamment en relation avec les figures 1A à 1G, 2A à 2B et 3.

De même, bien que l'invention s'applique plus particulièrement à un langage de type objet, et plus particulièrement au "p-code" du langage "JAVA", obtenu après compilation, elle s'applique à un grand nombre de langage mettant en œuvre des données typées, tels les langages "ADA" ou "KAMEL" rappelés dans le préambule de la présente description.

Enfin, bien que l'invention soit particulièrement avantageuse pour des systèmes embarqués à puce électronique, dont les ressources

informatiques, tant de traitement de données que de stockage de ces données, sont limitées, notamment pour des cartes à puce, elle convient parfaitement, *a fortiori*, pour des systèmes plus puissants.

5

TABLE I

Préfixe	Type	Code
i	"*Int*"	000
s	"*Short*"	001
b	"*Byte*"	010
z	"*Boolean*"	011
a	"*Object Reference*"	100

REVENDICATIONS

1. Procédé pour l'exécution sécurisée d'une séquence d'instructions d'une application informatique se présentant sous la forme de données typées enregistrées dans une première série d'emplacements déterminés d'une mémoire d'un système informatique, notamment un système embarqué à puce électronique, caractérisé en ce que des données supplémentaires dites éléments d'information de type sont associées à chacune desdites données typées, de manière à spécifier le type de ces données, en ce que lesdits éléments d'information de type sont enregistrés dans une deuxième série d'emplacements de mémoire déterminés (4, 5) de ladite mémoire (1) de système informatique (8), et en ce que, avant l'exécution d'instructions d'un type prédéterminé, il est procédé à une vérification en continu, préalable à l'exécution d'instructions prédéterminées, de la concordance entre un type indiqué par ces instructions et un type attendu indiqué par lesdits éléments d'information de type enregistrés dans ladite deuxième série d'emplacement de mémoire (4, 5), de manière n'autoriser ladite exécution qu'en cas de concordance entre lesdits types.

2. Procédé selon la revendication 1, caractérisé en ce que chacun desdits éléments d'information de type est constitué par une suite de bits enregistrés dans des emplacements de mémoire de ladite deuxième série (4, 5), en correspondance biunivoque avec des emplacements de mémoire de ladite première série (2, 3) dans lesquels sont enregistrées desdites données typées associées, et dont la configuration est représentative d'un desdits types de données typées.

3. Procédé selon la revendication 1, caractérisé en ce que lesdites instructions étant celles d'une application écrite en langage "JAVA" (marque déposée), lesdites données typées sont constituées par des

objets typés, en ce que ledit système informatique intègre une pièce de logicielle dite machine virtuelle "JAVA" (5) manipulant lesdits objets typés, en ce que, lesdits emplacements de mémoire (2-5) de ladite mémoire (1) du système informatique (8) étant organisés en piles comportant un nombre maximum de niveaux déterminé, chaque niveau constituant un desdits emplacements de mémoire, lesdits objets typés sont enregistrés dans au moins une première pile élémentaire dite zone de données (2) et un deuxième pile élémentaire dite zone de variables locales (3), et en ce que lesdits éléments d'information de type sont répartis dans deux piles élémentaires supplémentaires (4, 5) en relation biunivoque avec lesdites première (2) et deuxième (3) piles élémentaires, de manière à spécifier le type desdits objets associés enregistrés dans lesdites zones de données (2) et de variables locales (3).

4. Procédé selon la revendication 1, caractérisé en ce que lorsque ladite concordance n'est pas réalisée, l'exécution de ladite séquence d'instructions est interrompue et remplacée par l'exécution d'instructions correspondant à des mesures sécuritaires pré-programmées .

5. Procédé selon la revendication 3, caractérisé en ce que lesdits éléments d'information de type sont associés à des éléments d'information supplémentaires déterminant la taille desdits emplacements de mémoires desdites piles (2, 3) enregistrant lesdits objets typés, de manière à rendre variable la taille desdites piles, en fonction desdits objets à manipuler.

6. Procédé selon la revendication 3, caractérisé en ce que lesdits éléments d'information de type sont associés à des éléments d'information supplémentaires, dits drapeaux, de manière à marquer lesdits objets qui leur sont associés et à indiquer s'ils doivent être conservés dans lesdites piles (2, 3) ou peuvent être effacés.

7. Système embarqué à carte à puce électronique comprenant des moyens de traitement informatique de données et des moyens de mémoire pour l'exécution sécurisée d'une séquence d'instructions d'une application informatique se présentant sous la forme de données typées enregistrées dans une première série d'emplacements déterminés d'une mémoire d'un système informatique, caractérisé en ce que lesdits moyens de mémoire (1) comprennent une deuxième série d'emplacements déterminés (4, 5) pour l'enregistrement de données supplémentaires dites éléments d'information de type, associés à chacune desdites données typées, de manière à spécifier le type de ces données, et des moyens de vérification (6) permettant une vérification en continu, préalable à l'exécution d'instructions prédéterminées, de la concordance entre un type indiqué par ces instructions et un type indiqué par lesdits éléments d'information de type, de manière n'autoriser ladite exécution qu'en cas de concordance entre lesdits types.

8. Système selon la revendication 7, caractérisé en ce que, ladite première série d'emplacements déterminés de ladite mémoire (1) du système embarqué à puce électronique (8) étant organisée en piles comportant un nombre maximum de niveaux déterminé, chaque niveau constituant un desdits emplacements de mémoire, lesdites données typées sont enregistrées dans au moins une première pile élémentaire dite zone de données (2) et une deuxième pile élémentaire dite zone de variables locales (3), et en ce que ladite deuxième série d'emplacements de mémoire est aussi organisée en piles élémentaires (4, 5), en relation biunivoque avec lesdites première (2) et deuxième (3) piles élémentaires.

9. Système selon la revendication 8, caractérisé en ce que lesdits éléments d'information de type enregistrés dans ladite deuxième série d'emplacements de mémoire (4, 5) sont associés à des éléments d'information supplémentaires déterminant la taille desdits emplacements de mémoires desdites piles (2, 3) enregistrant lesdites données typées.

10. Système selon la revendication 7, caractérisé en ce que ledit système embarqué est une carte à puce (8).

BREVET D'INVENTION

PROCEDE DE SECURISATION D'UN LANGAGE DU TYPE A DONNEES TYPEES, NOTAMMENT DANS UN SYSTEME EMBARQUE ET SYSTEME EMBARQUE DE MISE EN ŒUVRE DU PROCEDE

Inventeurs : FOUGEROUX Nicolas, HAMEAU Patrice et LANDIER Olivier

Déposant : BULL CP8

ABREGE

L'invention concerne un procédé et un système embarqué à puce électronique (8) pour l'exécution sécurisée d'une séquence d'instructions d'une application informatique se présentant sous la forme d'objets ou de données typées, notamment écrite en langage "JAVA". La mémoire (1) est organisée en une première série de piles élémentaires (2, 3) pour l'enregistrement des instructions. On associe à chaque donnée ou objet typé un ou plusieurs bits dits de typage spécifiant le type. Ces bits sont enregistrés dans une deuxième série de piles élémentaires (4, 5), en relation biunivoque avec les piles (2, 3) de la première série. Avant l'exécution d'instructions de types prédéterminés, il est procédé à une vérification en continu, préalable à l'exécution de ces instructions, de la concordance entre un type indiqué par celles-ci et un type attendu, indiqué par les bits de typage. En cas de non-concordance l'exécution et stoppée.

FIGURE 3

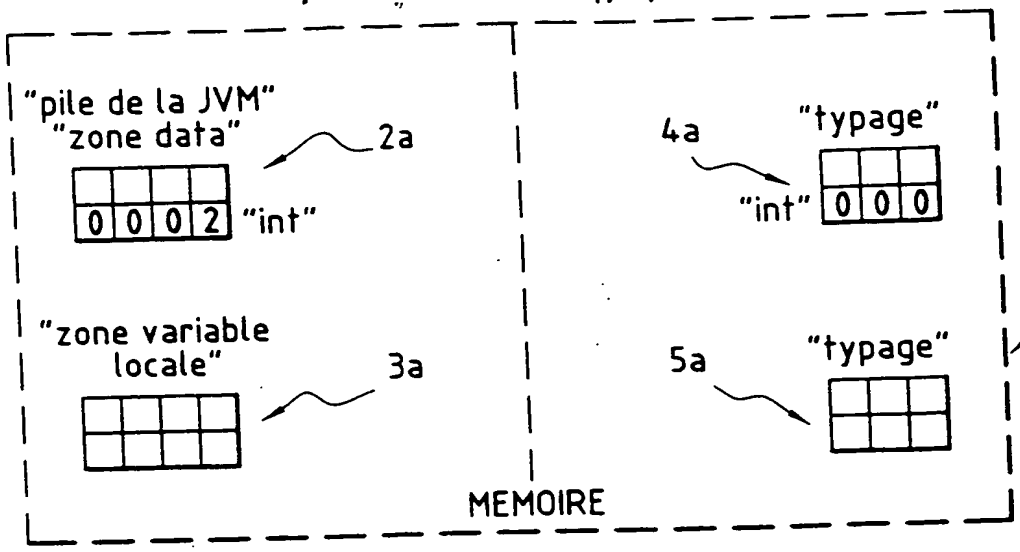


FIG.1A

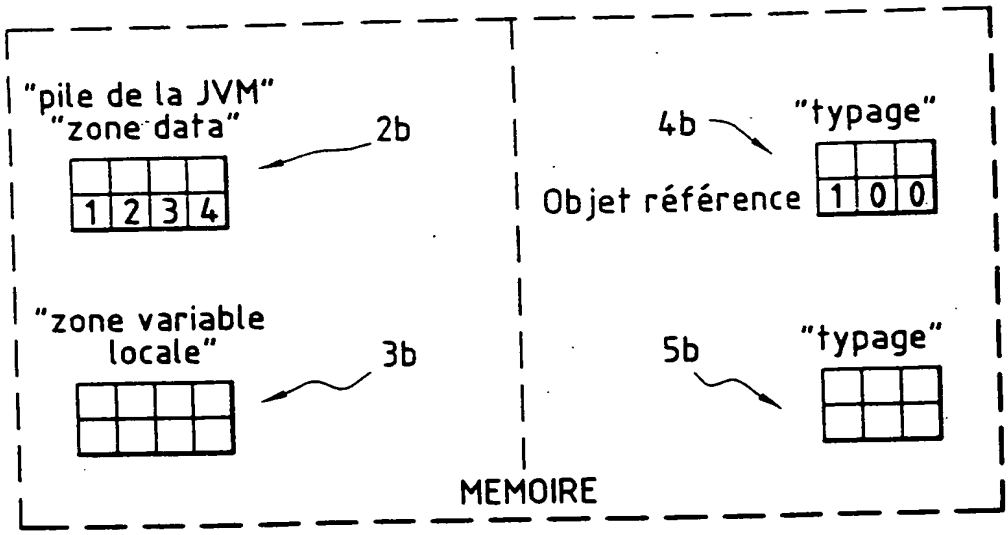


FIG.1B

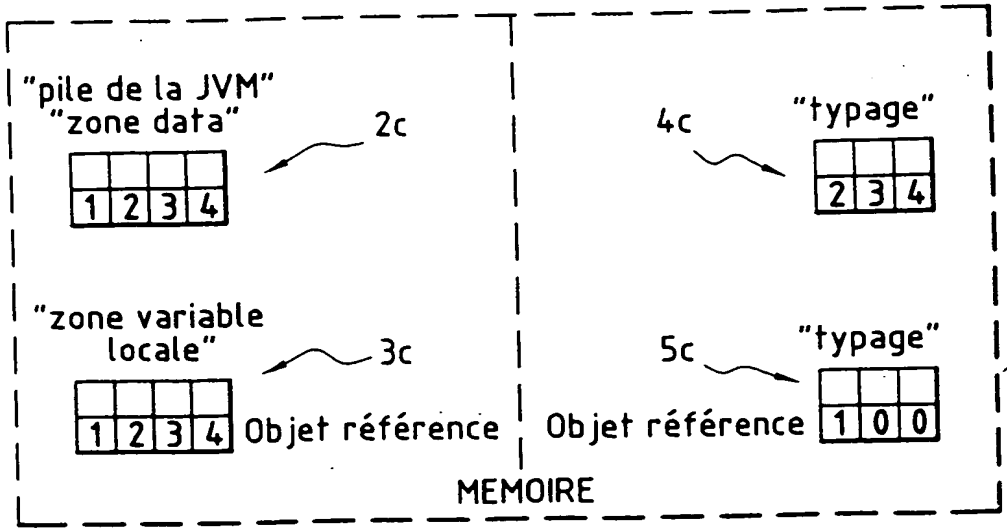


FIG.1C

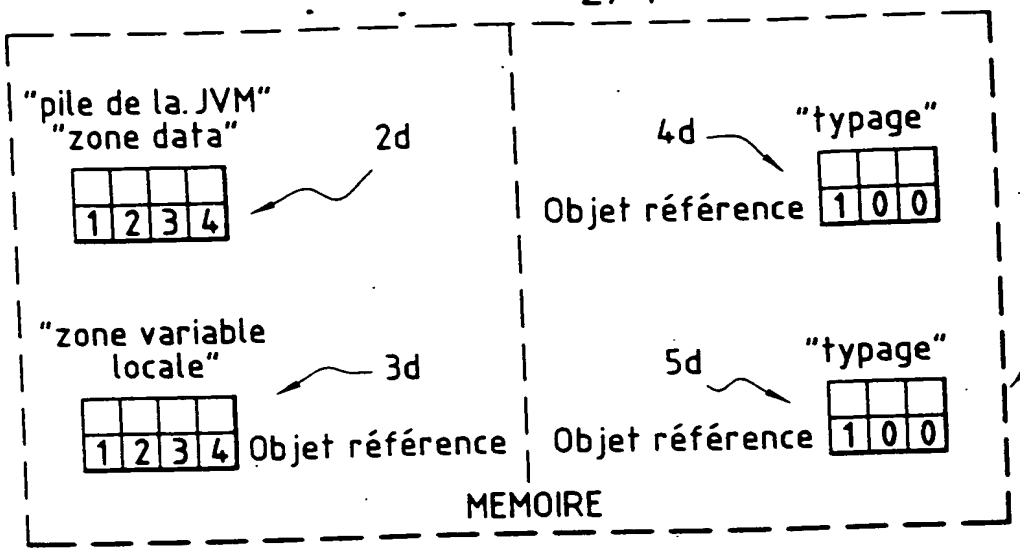


FIG.1D

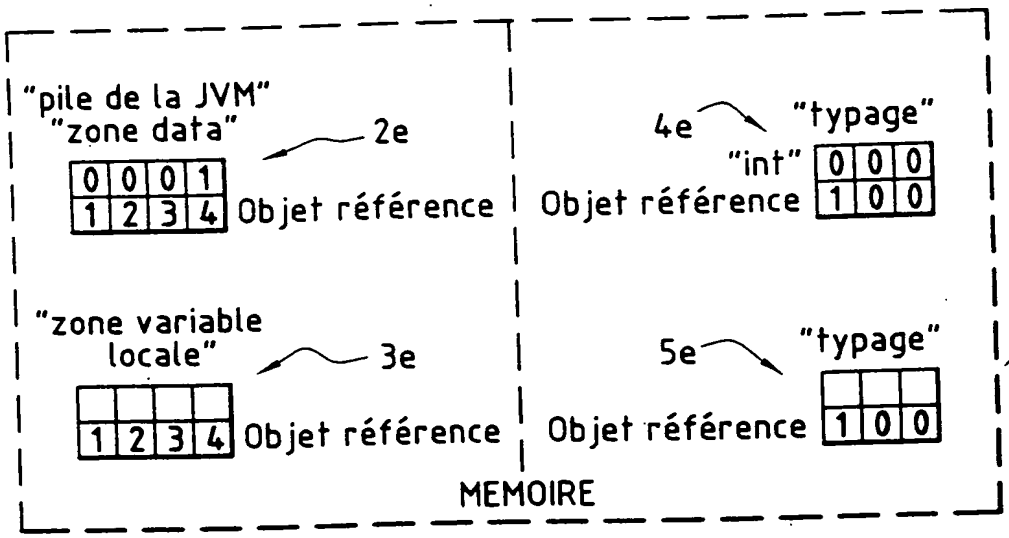


FIG.1E

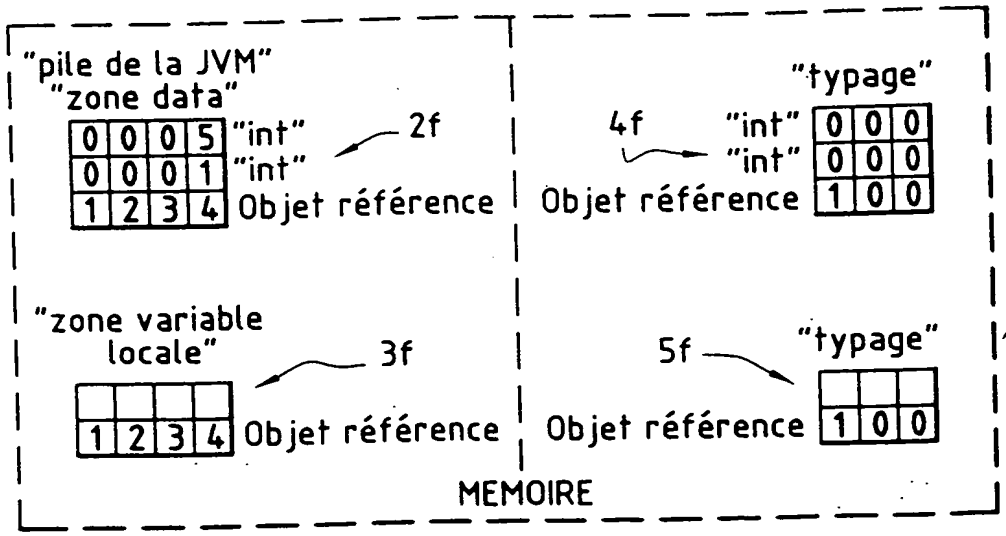


FIG.1F

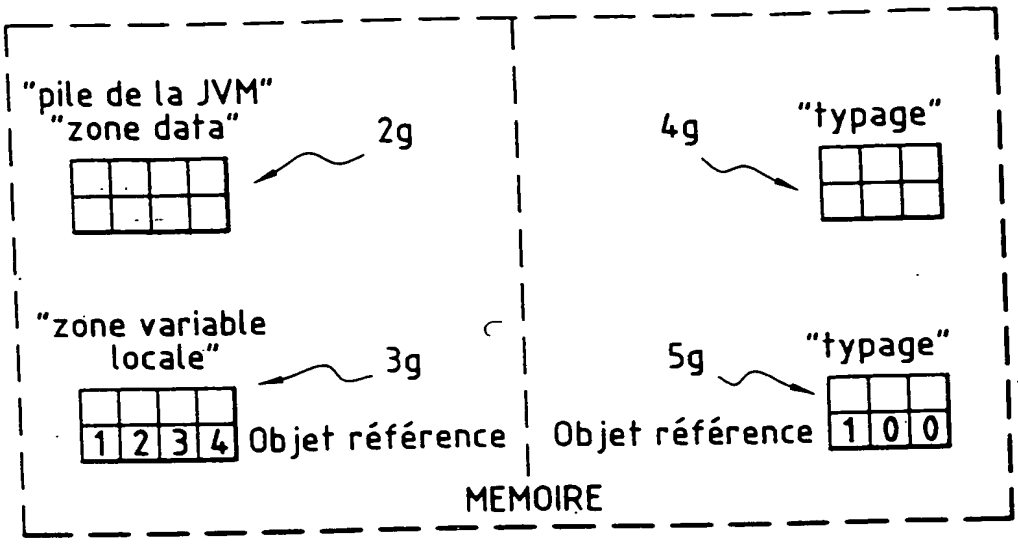


FIG.1G

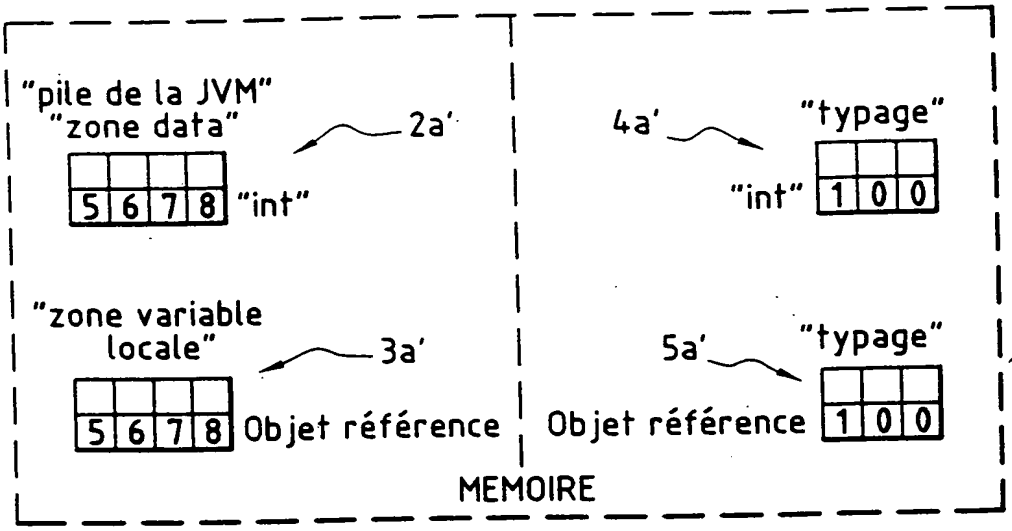


FIG.2A

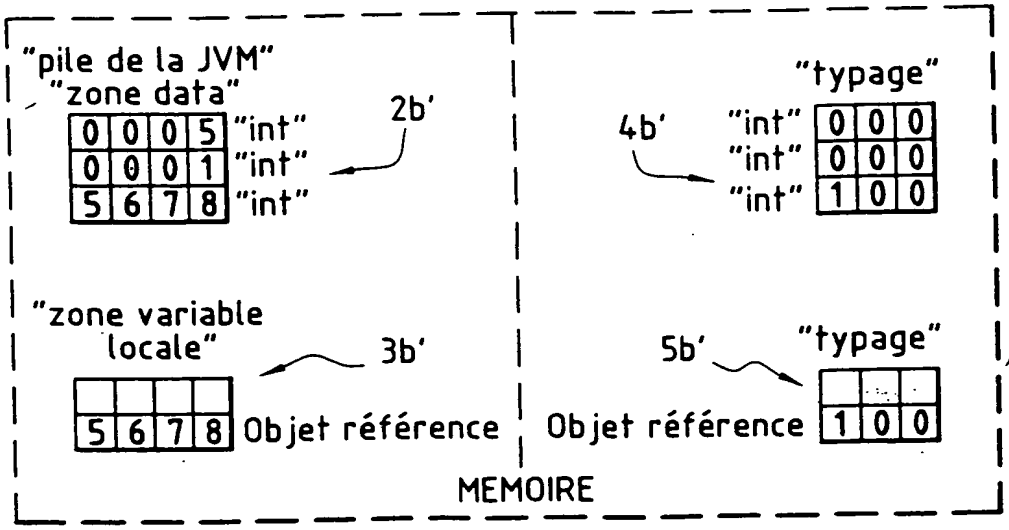


FIG.2B

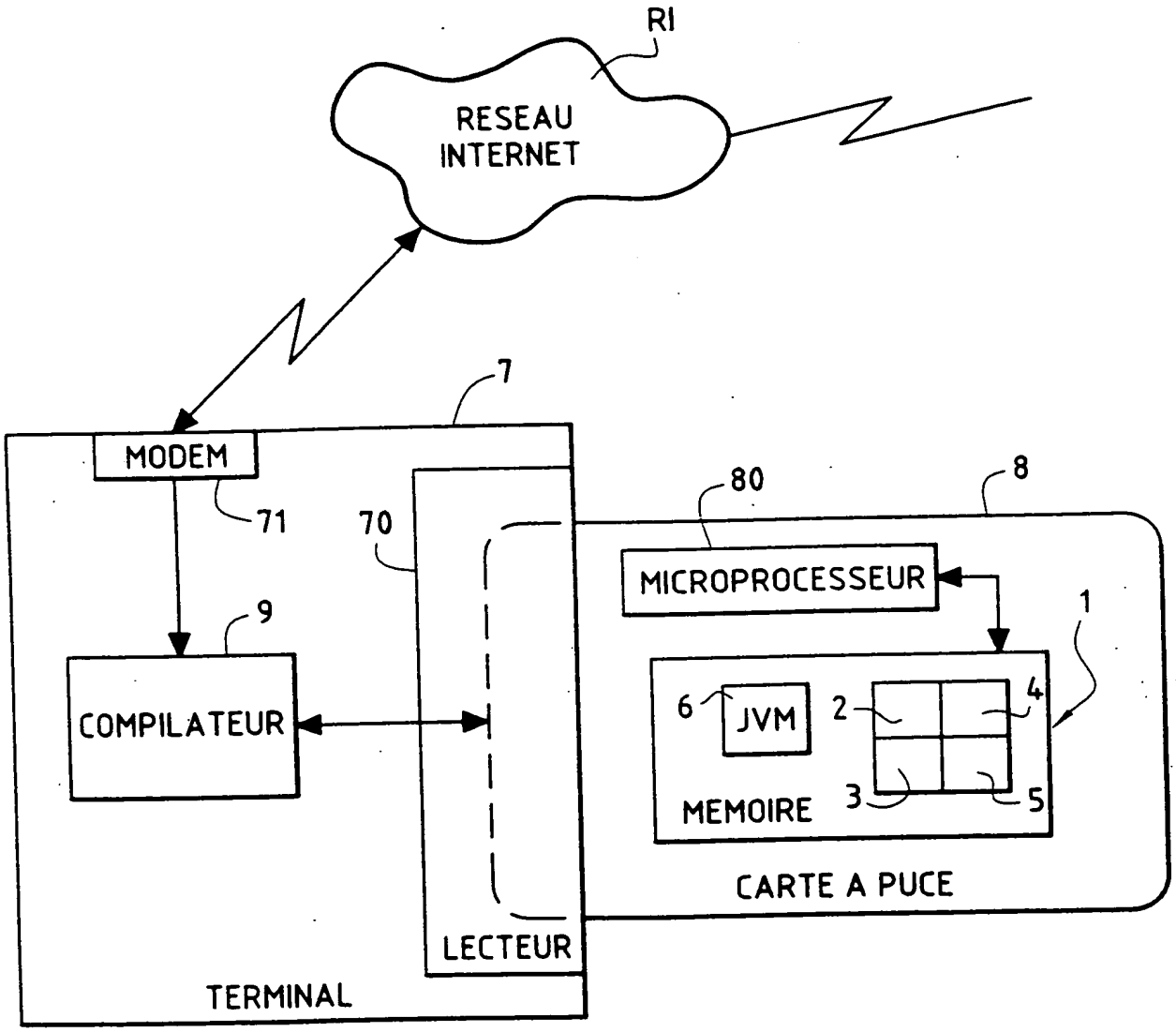

RI
RESEAU INTERNET
7
MODEM
71
70
80
8
MICROPROCESSEUR
1
9
COMPILATEUR
6
JVM
2
4
3
5
MEMOIRE
CARTE A PUCE
LECTEUR
TERMINAL

FIG.3